CS634 Midterm Project

Name: Adit Nuwal

UCID: an238

Email: an238@njit.edu

Table of Contents

Introduction

Association rule mining is a machine learning technique designed to uncover associations between categorical variables in data. In this project, we explore two different approaches to association rule mining: the Apriori Algorithm and the Brute Force Method. Our goal is to identify frequently occurring item combinations and generate meaningful association rules using a simulated dataset that mimics real-world retail transactions.

This project involves:

- Creating multiple transaction datasets using a predefined list of grocery items.
- Implementing both Apriori and Brute Force approaches to extract frequent itemsets and generate association rules.
- Comparing the execution times of both methods to evaluate computational efficiency.
- Allowing users to customize **minimum support** and **confidence** thresholds to refine the output.

We analyze these techniques to show why Apriori is the preferred choice for handling large-scale datasets, making it a crucial tool in data analytics and business intelligence.

Project Description

The project has two parts:

1. **Transaction Database Creation:**
    - Generate 30 grocery items commonly seen in supermarkets.
    - Create 5 different databases, each containing 20 transactions made with random selections from 30 general items.
2. **Association Rule Mining:**
    - Implement the Apriori algorithm to discover frequent itemsets and generate association rules.
    - Implement the Brute Force method to enumerate all possible itemsets and determine frequent itemsets.

  - Compare the execution times of both methods to demonstrate the efficiency of Apriori over the brute force approach.

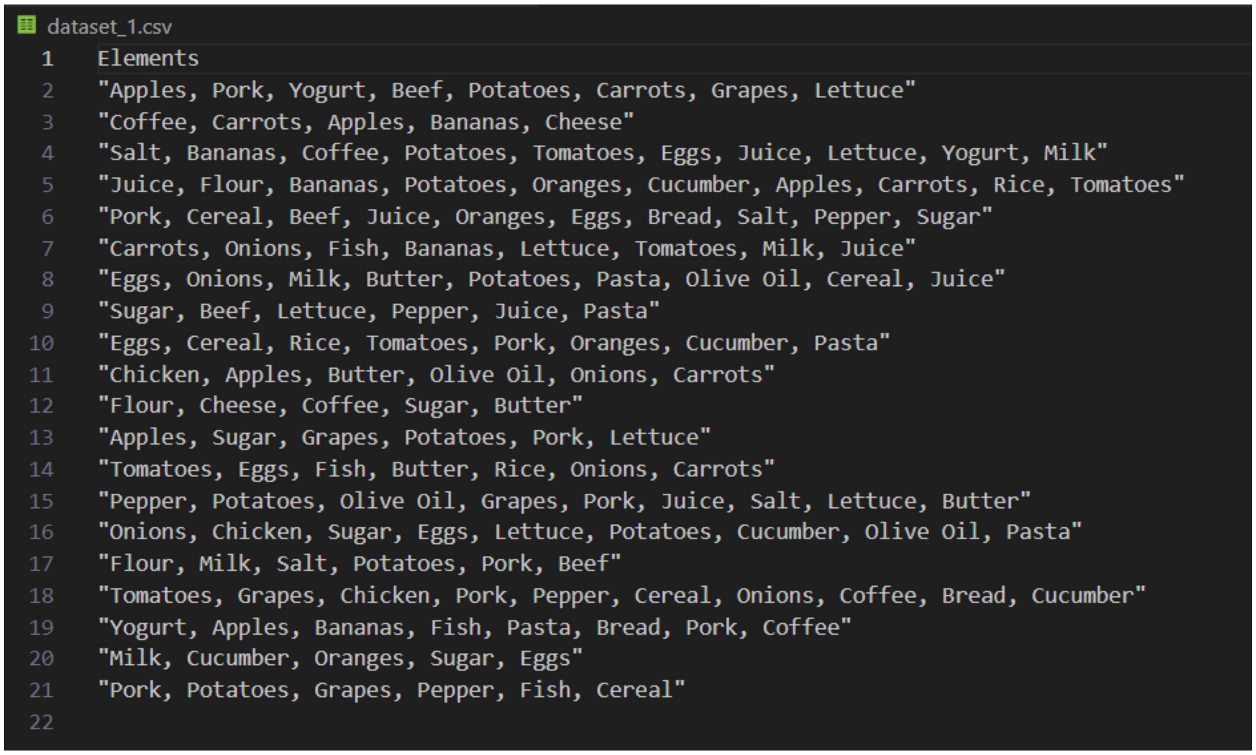
Summary

- Users specify minimum support and confidence values.
- Five transaction datasets are generated and stored.
- Both Apriori and Brute Force methods are applied to mine frequent itemsets and generate association rules.
- The results include execution times, frequent itemsets, and the strongest association rules.

Source Code

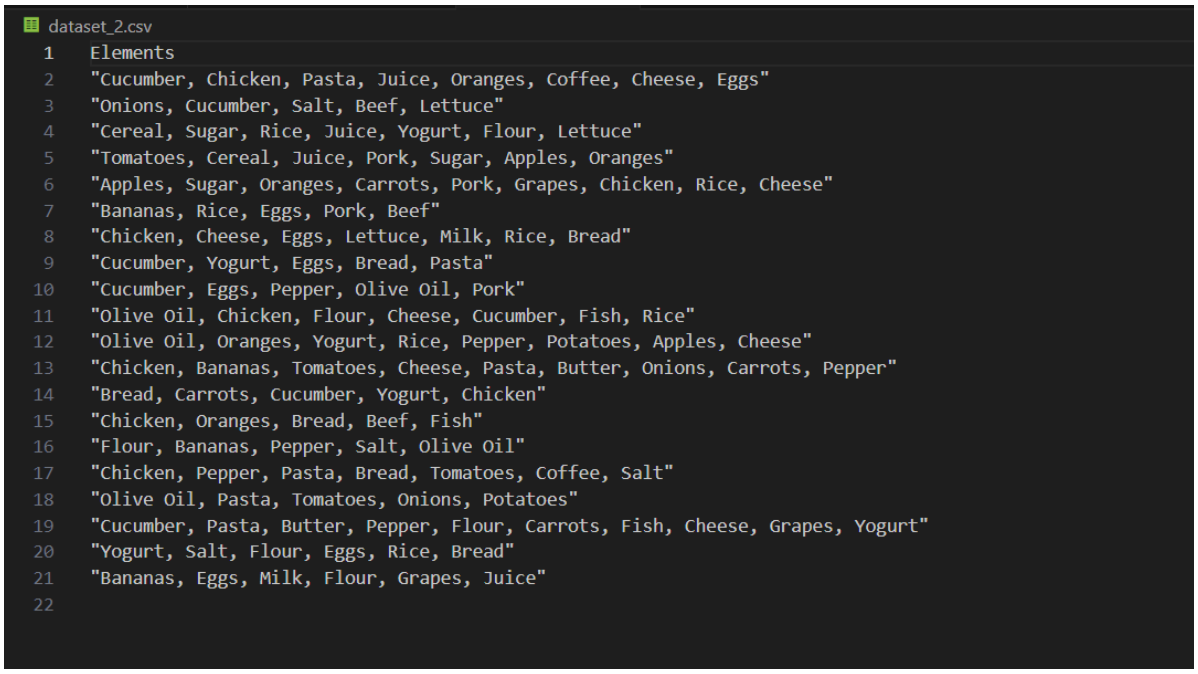
```
import os
import pandas as pd
import itertools
import time
import random
from collections import defaultdict

class DataProcessor:
    def __init__(self, items):
        self.dataItems = items

    def createDataSets(self, numSets=20):
        return [random.sample(self.dataItems, random.randint(5, 10)) for _
in range(numSets)]

    def storeData(self, fileData, fileName):
        df = pd.DataFrame({"Elements": [", ".join(entry) for entry in
fileData]})
        df.to_csv(fileName, index=False)

    def retrieveData(self, fileName):
        df = pd.read_csv(fileName)
        return [set(items.split(", ")) for items in df["Elements"]]

class AssociationRuleMiner:
    def __init__(self, minSupp, minConf):
        self.minSupp = minSupp
        self.minConf = minConf

    def analyzeApriori(self, fileData):
        maxLength = len(max(fileData, key=len))
        freqSets = []
        uniqueItems = set().union(*fileData)
        itemCounts = defaultdict(int)

        for item in uniqueItems:
            elementSet = frozenset([item])
            itemCounts[elementSet] = sum(1 for group in fileData if
elementSet.issubset(group))

        activeSets = [elementSet for elementSet, count in
itemCounts.items() if count/len(fileData) >= self.minSupp]
        freqSets.extend(activeSets)
        if not activeSets:
            return [], []

        level = 2
        while level <= maxLength:
            newCombos = {a.union(b) for i, a in enumerate(activeSets) for
b in activeSets[i+1:] if tuple(sorted(a))[:-1] == tuple(sorted(b))[:-1]
and len(a.union(b)) == level}

            entryCounts = defaultdict(int)
            for combo in newCombos:
                for group in fileData:
                    if combo.issubset(group):
                        entryCounts[combo] += 1
            activeSets = [elementSet for elementSet, count in
entryCounts.items() if count/len(fileData) >= self.minSupp]
            if not activeSets:
                break
            freqSets.extend(activeSets)
            level += 1

        return freqSets, self._generateRules(fileData, freqSets)

    def analyzeBruteForce(self, fileData):
        uniqueElements = set().union(*fileData)
        freqSets = []

        for size in range(1, min(5, len(uniqueElements)+1)):
            combinations = itertools.combinations(uniqueElements, size)
            entryCounts = {frozenset(combo): sum(1 for group in fileData
if frozenset(combo).issubset(group)) for combo in combinations}
            activeSets = [elementSet for elementSet, count in
entryCounts.items() if count/len(fileData) >= self.minSupp]
            freqSets.extend(activeSets)

        return freqSets, self._generateRules(fileData, freqSets)

    def _generateRules(self, fileData, freqSets):
        rulesList = []
        for itemset in freqSets:
            if len(itemset) < 2:
                continue
            for subset in itertools.combinations(itemset, len(itemset)-1):
                subset = frozenset(subset)
                remaining = itemset - subset
                subsetCount = sum(1 for group in fileData if
subset.issubset(group))
                fullCount = sum(1 for group in fileData if
itemset.issubset(group))
                if subsetCount == 0:
                    continue
                confValue = fullCount / subsetCount
                if confValue >= self.minConf:
                    suppValue = fullCount / len(fileData)
                    rulesList.append((subset, remaining, suppValue,
confValue))
        return rulesList

def runAnalysis():
    print("\nCS634 Midterm Project")
    print("Name: Adit Nuwal")
    print("UCID: an238")
    print("Email: an238@njit.edu\n")

    minSupp = float(input("Enter the minimum support value (between 0 and
1): "))
    minConf = float(input("Enter the minimum confidence value (between 0
and 1): "))

    dataItems = ["Milk", "Bread", "Eggs", "Cheese", "Butter", "Yogurt",
"Chicken", "Beef", "Pork", "Fish",
                 "Apples", "Bananas", "Grapes", "Oranges", "Tomatoes",
"Potatoes", "Onions", "Carrots", "Lettuce", "Cucumber",
                 "Rice", "Pasta", "Flour", "Sugar", "Salt", "Pepper",
"Olive Oil", "Cereal", "Juice", "Coffee"]

    processor = DataProcessor(dataItems)
    miner = AssociationRuleMiner(minSupp, minConf)
    fileList = []

    for i in range(1, 6):
        dataset = processor.createDataSets()
        fileName = f"dataset_{i}.csv"
        processor.storeData(dataset, fileName)
        fileList.append(fileName)
    for idx, fileName in enumerate(fileList, 1):
        print(f"\nProcessing dataset {idx}: {fileName}")
        fileData = processor.retrieveData(fileName)

        startTime = time.perf_counter()
        aprioriSets, aprioriRules = miner.analyzeApriori(fileData)
        aprioriExecTime = time.perf_counter() - startTime

        startTime = time.perf_counter()
        bruteSets, bruteRules = miner.analyzeBruteForce(fileData)
        bruteExecTime = time.perf_counter() - startTime

        print(f"\nApriori algorithm took {aprioriExecTime:.8f} seconds.")
        print(f"Brute-force algorithm took {bruteExecTime:.8f} seconds.")
        print(f"We found {len(aprioriSets)} frequent itemsets.")
        print(f"Generated {len(aprioriRules)} association rules.")

        if aprioriRules:
            print("\nHere are the top 5 strongest association rules:")
            sortedRules = sorted(aprioriRules, key=lambda x: (-x[3],
-x[2]))
            for rule in sortedRules[:5]:
                print(f"{set(rule[0])} => {set(rule[1])} | Confidence:
{rule[3]:.2f}, Support: {rule[2]:.2f}")

if __name__ == "__main__":
    runAnalysis()
```

Dataset Screenshots



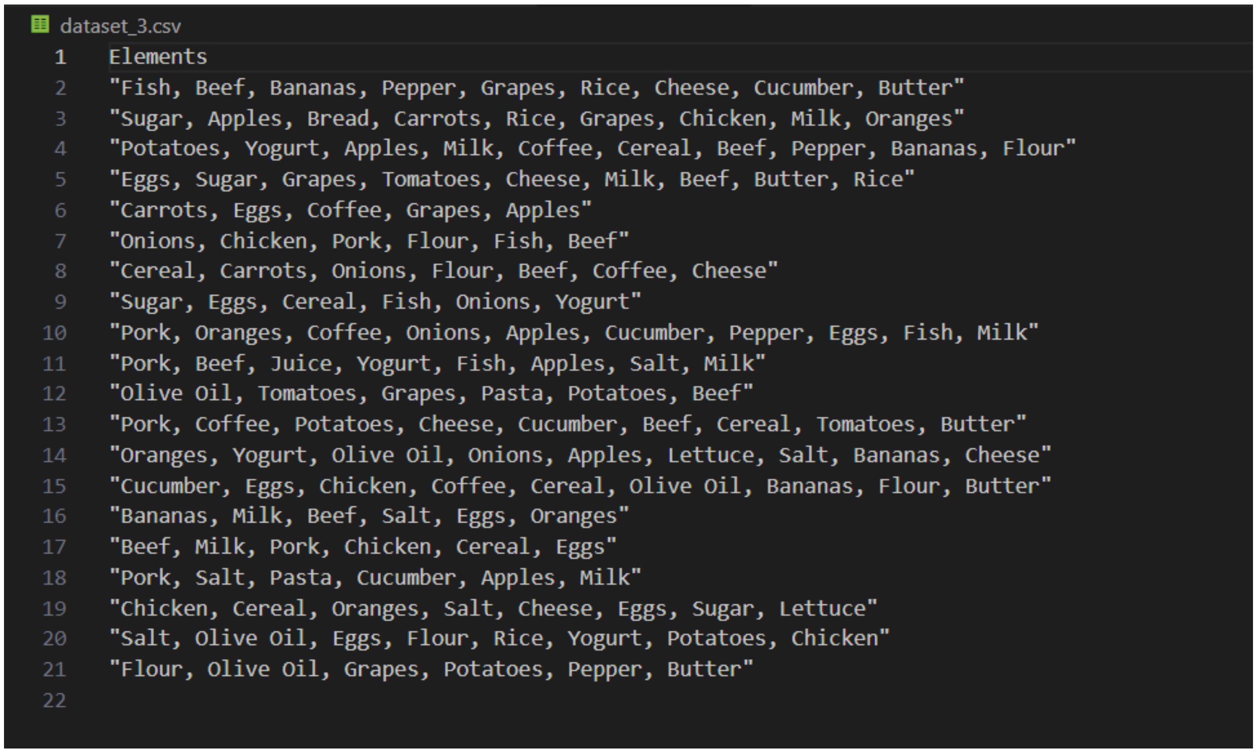
dataset_1.csv

```
Elements
"Apples, Pork, Yogurt, Beef, Potatoes, Carrots, Grapes, Lettuce"
"Coffee, Carrots, Apples, Bananas, Cheese"
"Salt, Bananas, Coffee, Potatoes, Tomatoes, Eggs, Juice, Lettuce, Yogurt, Milk"
"Juice, Flour, Bananas, Potatoes, Oranges, Cucumber, Apples, Carrots, Rice, Tomatoes"
"Pork, Cereal, Beef, Juice, Oranges, Eggs, Bread, Salt, Pepper, Sugar"
"Carrots, Onions, Fish, Bananas, Lettuce, Tomatoes, Milk, Juice"
"Eggs, Onions, Milk, Butter, Potatoes, Pasta, Olive Oil, Cereal, Juice"
"Sugar, Beef, Lettuce, Pepper, Juice, Pasta"
"Eggs, Cereal, Rice, Tomatoes, Pork, Oranges, Cucumber, Pasta"
"Chicken, Apples, Butter, Olive Oil, Onions, Carrots"
"Flour, Cheese, Coffee, Sugar, Butter"
"Apples, Sugar, Grapes, Potatoes, Pork, Lettuce"
"Tomatoes, Eggs, Fish, Butter, Rice, Onions, Carrots"
"Pepper, Potatoes, Olive Oil, Grapes, Pork, Juice, Salt, Lettuce, Butter"
"Onions, Chicken, Sugar, Eggs, Lettuce, Potatoes, Cucumber, Olive Oil, Pasta"
"Flour, Milk, Salt, Potatoes, Pork, Beef"
"Tomatoes, Grapes, Chicken, Pork, Pepper, Cereal, Onions, Coffee, Bread, Cucumber"
"Yogurt, Apples, Bananas, Fish, Pasta, Bread, Pork, Coffee"
"Milk, Cucumber, Oranges, Sugar, Eggs"
"Pork, Potatoes, Grapes, Pepper, Fish, Cereal"

```

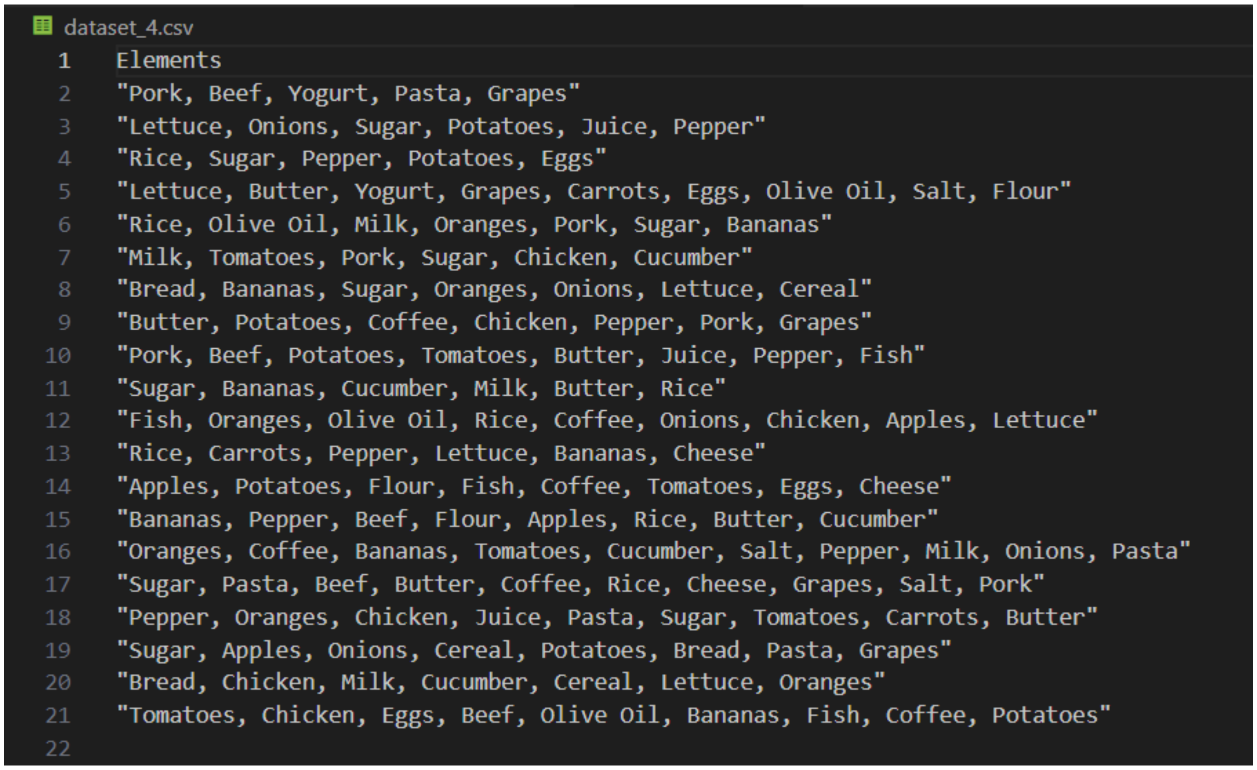


dataset_2.csv

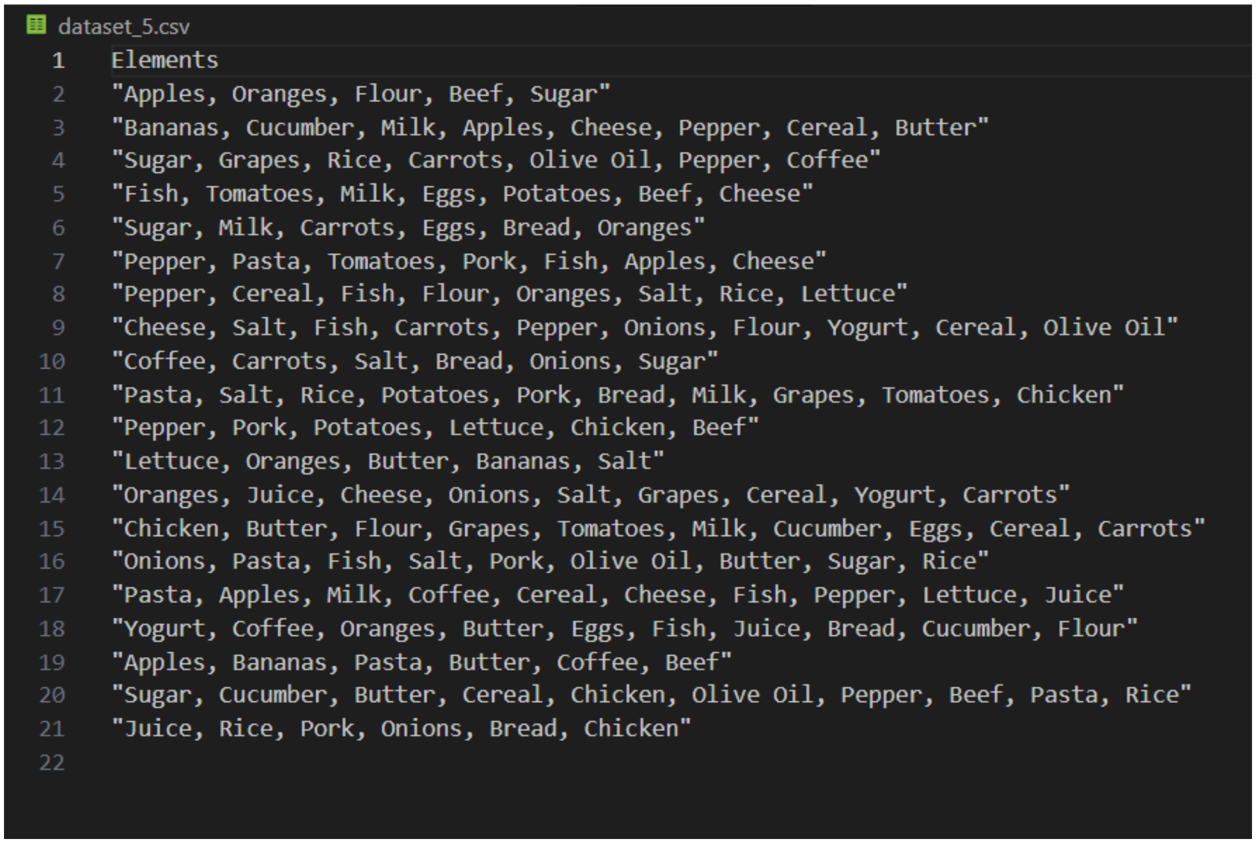
```
Elements
"Cucumber, Chicken, Pasta, Juice, Oranges, Coffee, Cheese, Eggs"
"Onions, Cucumber, Salt, Beef, Lettuce"
"Cereal, Sugar, Rice, Juice, Yogurt, Flour, Lettuce"
"Tomatoes, Cereal, Juice, Pork, Sugar, Apples, Oranges"
"Apples, Sugar, Oranges, Carrots, Pork, Grapes, Chicken, Rice, Cheese"
"Bananas, Rice, Eggs, Pork, Beef"
"Chicken, Cheese, Eggs, Lettuce, Milk, Rice, Bread"
"Cucumber, Yogurt, Eggs, Bread, Pasta"
"Cucumber, Eggs, Pepper, Olive Oil, Pork"
"Olive Oil, Chicken, Flour, Cheese, Cucumber, Fish, Rice"
"Olive Oil, Oranges, Yogurt, Rice, Pepper, Potatoes, Apples, Cheese"
"Chicken, Bananas, Tomatoes, Cheese, Pasta, Butter, Onions, Carrots, Pepper"
"Bread, Carrots, Cucumber, Yogurt, Chicken"
"Chicken, Oranges, Bread, Beef, Fish"
"Flour, Bananas, Pepper, Salt, Olive Oil"
"Chicken, Pepper, Pasta, Bread, Tomatoes, Coffee, Salt"
"Olive Oil, Pasta, Tomatoes, Onions, Potatoes"
"Cucumber, Pasta, Butter, Pepper, Flour, Carrots, Fish, Cheese, Grapes, Yogurt"
"Yogurt, Salt, Flour, Eggs, Rice, Bread"
"Bananas, Eggs, Milk, Flour, Grapes, Juice"

```


dataset_3.csv

```
Elements
"Fish, Beef, Bananas, Pepper, Grapes, Rice, Cheese, Cucumber, Butter"
"Sugar, Apples, Bread, Carrots, Rice, Grapes, Chicken, Milk, Oranges"
"Potatoes, Yogurt, Apples, Milk, Coffee, Cereal, Beef, Pepper, Bananas, Flour"
"Eggs, Sugar, Grapes, Tomatoes, Cheese, Milk, Beef, Butter, Rice"
"Carrots, Eggs, Coffee, Grapes, Apples"
"Onions, Chicken, Pork, Flour, Fish, Beef"
"Cereal, Carrots, Onions, Flour, Beef, Coffee, Cheese"
"Sugar, Eggs, Cereal, Fish, Onions, Yogurt"
"Pork, Oranges, Coffee, Onions, Apples, Cucumber, Pepper, Eggs, Fish, Milk"
"Pork, Beef, Juice, Yogurt, Fish, Apples, Salt, Milk"
"Olive Oil, Tomatoes, Grapes, Pasta, Potatoes, Beef"
"Pork, Coffee, Potatoes, Cheese, Cucumber, Beef, Cereal, Tomatoes, Butter"
"Oranges, Yogurt, Olive Oil, Onions, Apples, Lettuce, Salt, Bananas, Cheese"
"Cucumber, Eggs, Chicken, Coffee, Cereal, Olive Oil, Bananas, Flour, Butter"
"Bananas, Milk, Beef, Salt, Eggs, Oranges"
"Beef, Milk, Pork, Chicken, Cereal, Eggs"
"Pork, Salt, Pasta, Cucumber, Apples, Milk"
"Chicken, Cereal, Oranges, Salt, Cheese, Eggs, Sugar, Lettuce"
"Salt, Olive Oil, Eggs, Flour, Rice, Yogurt, Potatoes, Chicken"
"Flour, Olive Oil, Grapes, Potatoes, Pepper, Butter"

```


dataset_4.csv

```
Elements
"Pork, Beef, Yogurt, Pasta, Grapes"
"Lettuce, Onions, Sugar, Potatoes, Juice, Pepper"
"Rice, Sugar, Pepper, Potatoes, Eggs"
"Lettuce, Butter, Yogurt, Grapes, Carrots, Eggs, Olive Oil, Salt, Flour"
"Rice, Olive Oil, Milk, Oranges, Pork, Sugar, Bananas"
"Milk, Tomatoes, Pork, Sugar, Chicken, Cucumber"
"Bread, Bananas, Sugar, Oranges, Onions, Lettuce, Cereal"
"Butter, Potatoes, Coffee, Chicken, Pepper, Pork, Grapes"
"Pork, Beef, Potatoes, Tomatoes, Butter, Juice, Pepper, Fish"
"Sugar, Bananas, Cucumber, Milk, Butter, Rice"
"Fish, Oranges, Olive Oil, Rice, Coffee, Onions, Chicken, Apples, Lettuce"
"Rice, Carrots, Pepper, Lettuce, Bananas, Cheese"
"Apples, Potatoes, Flour, Fish, Coffee, Tomatoes, Eggs, Cheese"
"Bananas, Pepper, Beef, Flour, Apples, Rice, Butter, Cucumber"
"Oranges, Coffee, Bananas, Tomatoes, Cucumber, Salt, Pepper, Milk, Onions, Pasta"
"Sugar, Pasta, Beef, Butter, Coffee, Rice, Cheese, Grapes, Salt, Pork"
"Pepper, Oranges, Chicken, Juice, Pasta, Sugar, Tomatoes, Carrots, Butter"
"Sugar, Apples, Onions, Cereal, Potatoes, Bread, Pasta, Grapes"
"Bread, Chicken, Milk, Cucumber, Cereal, Lettuce, Oranges"
"Tomatoes, Chicken, Eggs, Beef, Olive Oil, Bananas, Fish, Coffee, Potatoes"

```



dataset_5.csv

```
Elements
"Apples, Oranges, Flour, Beef, Sugar"
"Bananas, Cucumber, Milk, Apples, Cheese, Pepper, Cereal, Butter"
"Sugar, Grapes, Rice, Carrots, Olive Oil, Pepper, Coffee"
"Fish, Tomatoes, Milk, Eggs, Potatoes, Beef, Cheese"
"Sugar, Milk, Carrots, Eggs, Bread, Oranges"
"Pepper, Pasta, Tomatoes, Pork, Fish, Apples, Cheese"
"Pepper, Cereal, Fish, Flour, Oranges, Salt, Rice, Lettuce"
"Cheese, Salt, Fish, Carrots, Pepper, Onions, Flour, Yogurt, Cereal, Olive Oil"
"Coffee, Carrots, Salt, Bread, Onions, Sugar"
"Pasta, Salt, Rice, Potatoes, Pork, Bread, Milk, Grapes, Tomatoes, Chicken"
"Pepper, Pork, Potatoes, Lettuce, Chicken, Beef"
"Lettuce, Oranges, Butter, Bananas, Salt"
"Oranges, Juice, Cheese, Onions, Salt, Grapes, Cereal, Yogurt, Carrots"
"Chicken, Butter, Flour, Grapes, Tomatoes, Milk, Cucumber, Eggs, Cereal, Carrots"
"Onions, Pasta, Fish, Salt, Pork, Olive Oil, Butter, Sugar, Rice"
"Pasta, Apples, Milk, Coffee, Cereal, Cheese, Fish, Pepper, Lettuce, Juice"
"Yogurt, Coffee, Oranges, Butter, Eggs, Fish, Juice, Bread, Cucumber, Flour"
"Apples, Bananas, Pasta, Butter, Coffee, Beef"
"Sugar, Cucumber, Butter, Cereal, Chicken, Olive Oil, Pepper, Beef, Pasta, Rice"
"Juice, Rice, Pork, Onions, Bread, Chicken"

```

Code Explanation

DataProcessor Class:

This class is responsible for handling the transactional datasets, including their creation, storage, and retrieval.

- createDataSets(numSets=20): Generates synthetic transaction datasets. Each dataset contains random selections of grocery items.
- storeData(fileData, fileName): Saves the generated transactions into a CSV file for later analysis.
- retrieveData(fileName): Reads transaction data from the CSV file and structures it as a list of sets.

AssociationRuleMiner Class:

This class implements both the Apriori and Brute Force methods for association rule mining.

Apriori Algorithm (analyzeApriori)

- Step 1: Identify frequent 1-itemsets based on the given support threshold.
- Step 2: Generate larger itemsets iteratively, pruning the search space to reduce computation.
- Step 3: Extract association rules using confidence threshold and store them.
- Optimization: The Apriori method reduces the number of candidate itemsets significantly by leveraging prior knowledge of frequent itemsets.

Brute Force Method (analyzeBruteForce)

- Step 1: Enumerates all possible 1-itemsets, 2-itemsets, and higher.
- Step 2: Check each combination for frequency based on the support threshold.
- Step 3: Generates association rules in a computationally expensive manner.
- Limitations: The brute force method is inefficient as it does not eliminate unnecessary itemsets early in the process.

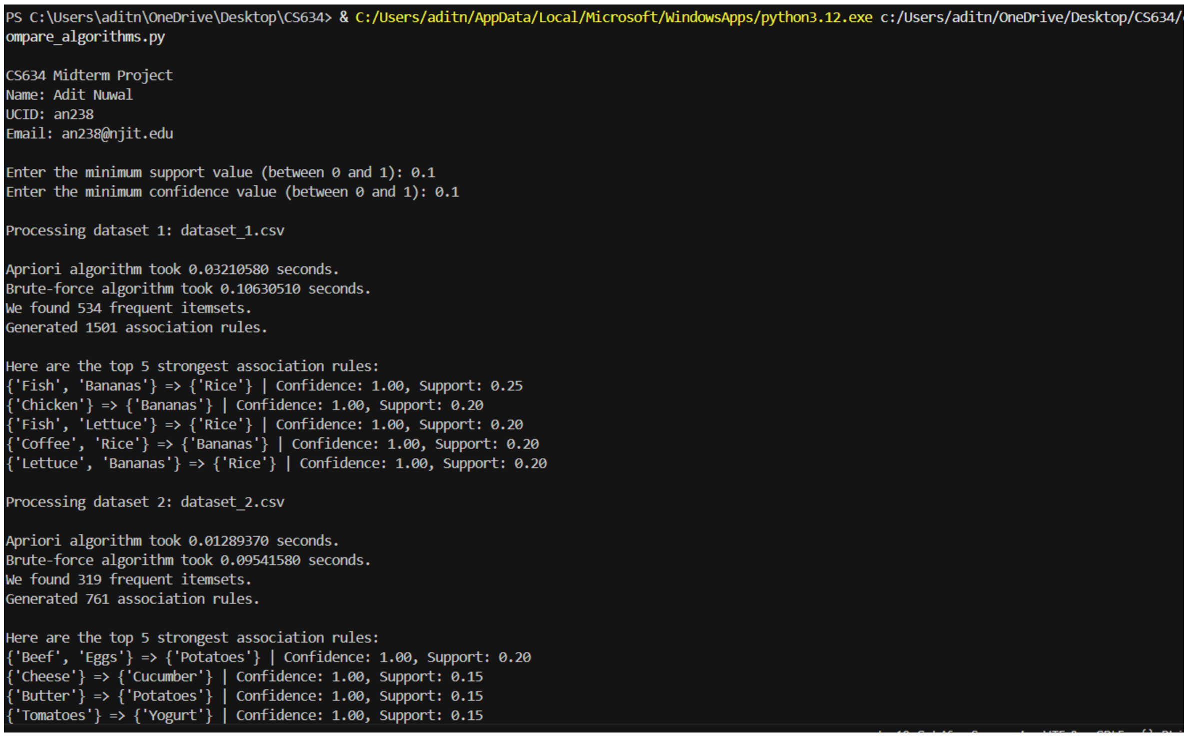
Rule Generation (_generateRules)

- Extracts meaningful association rules from frequent itemsets.
- Computes confidence for each rule and retains only those exceeding the minimum confidence threshold.

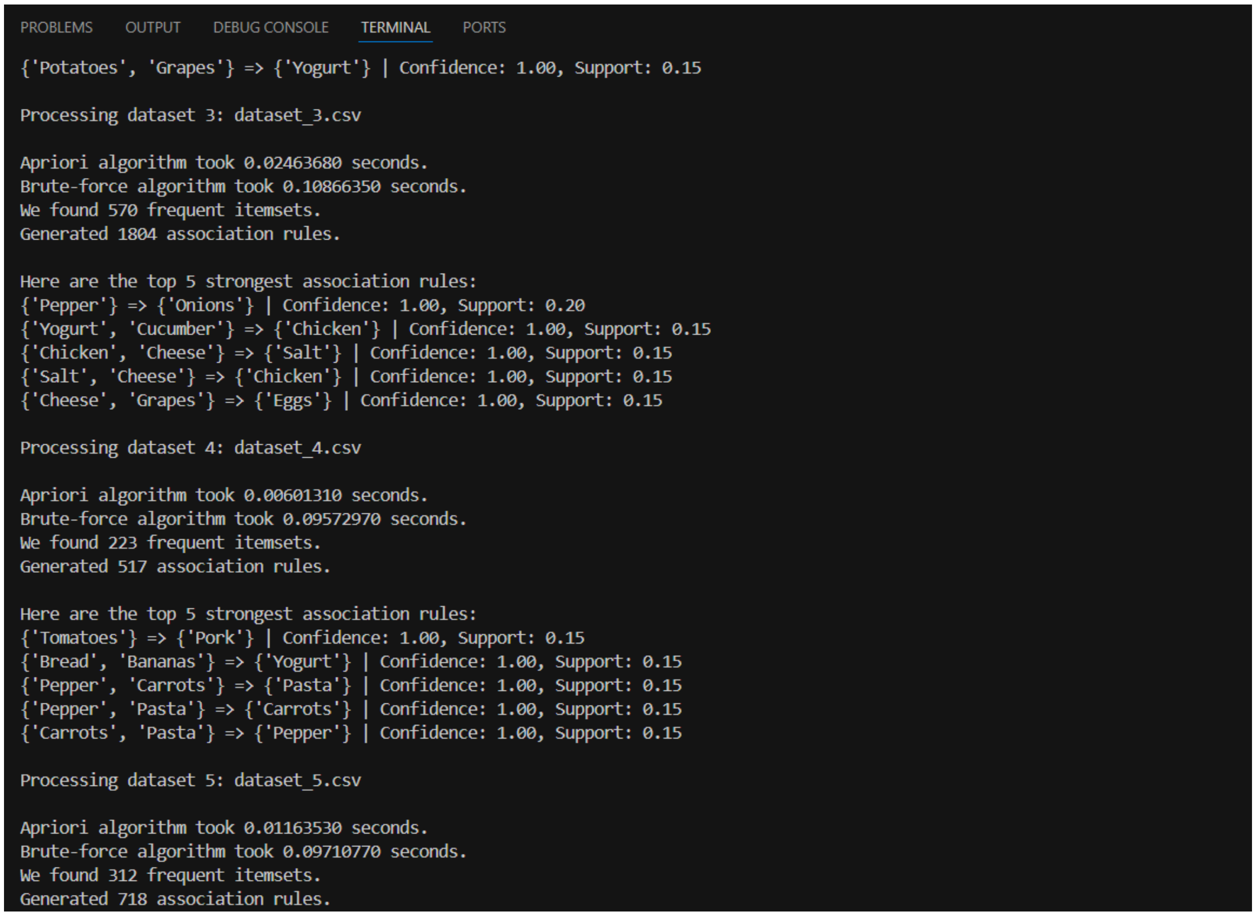
runAnalysis Function:

- Prompts the user for minimum support and confidence values.
- Generates and stores five different transaction datasets.
- Executes both Apriori and Brute Force methods on each dataset.
- Measures and displays execution time and extracted rules.

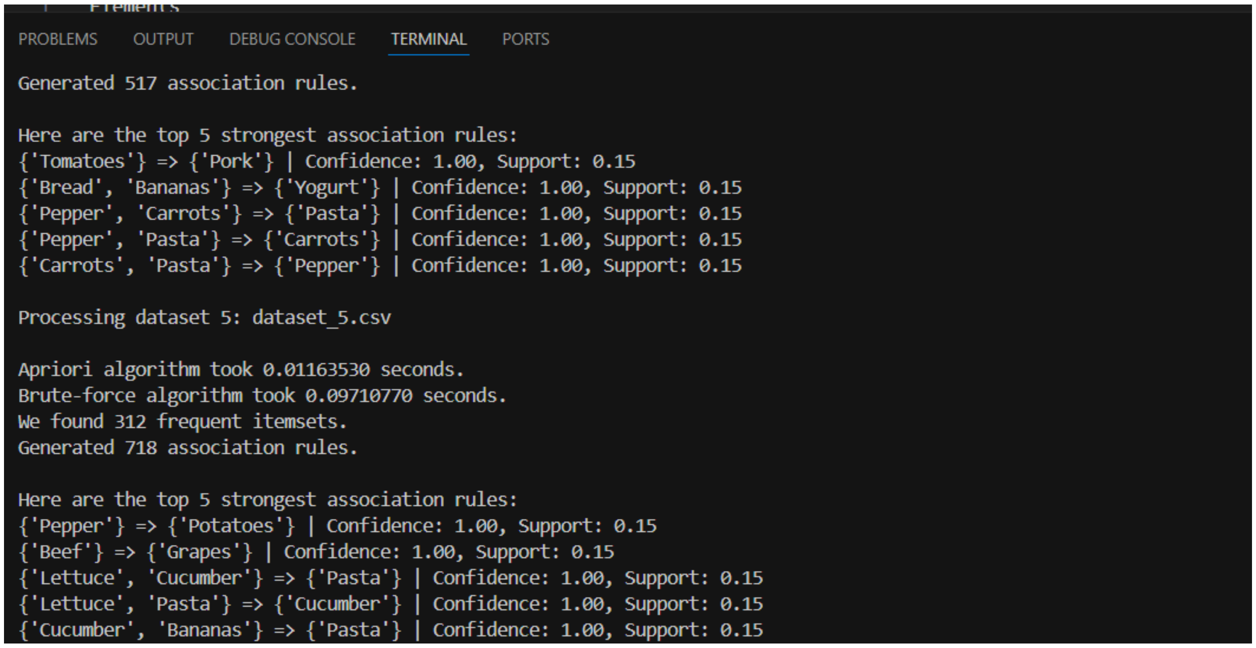
Output



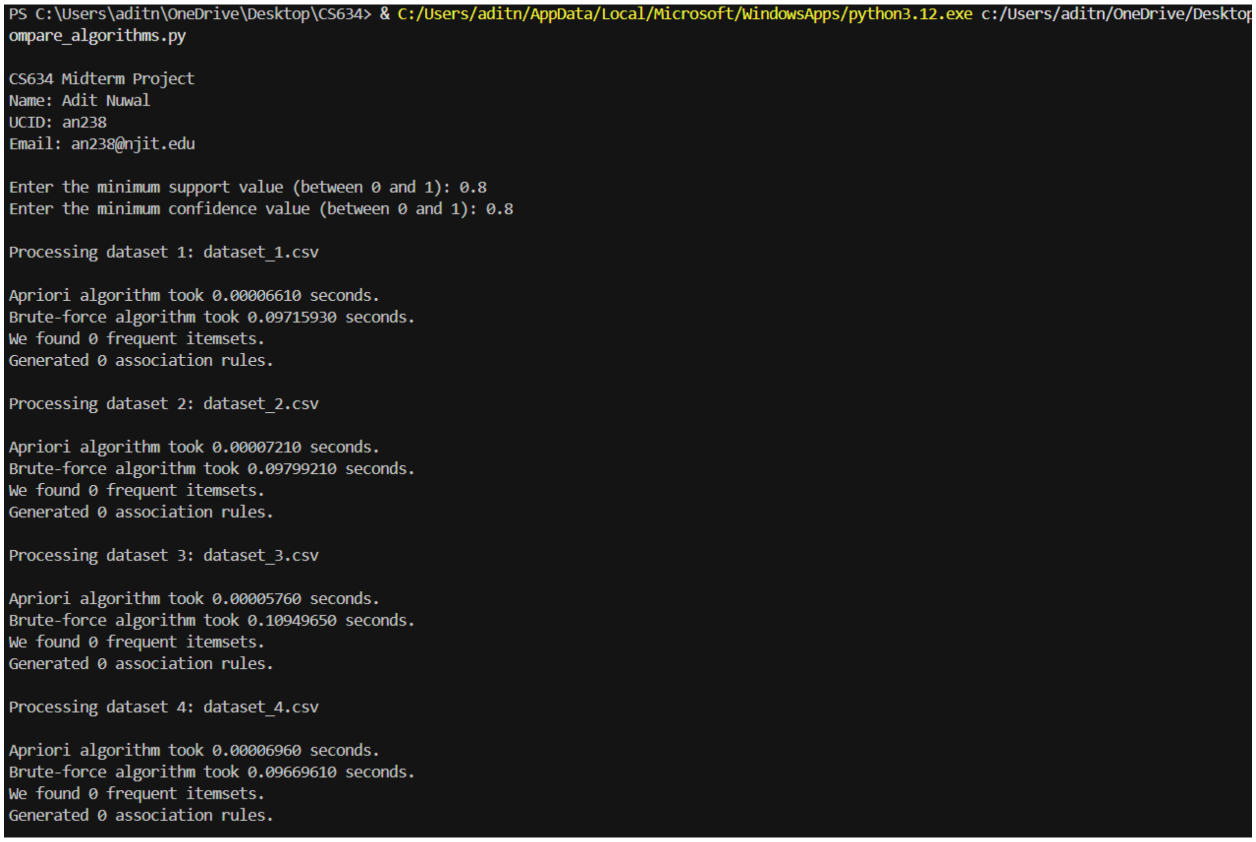
```
PS C:\Users\aditn\OneDrive\Desktop\CS634> & C:/Users/aditn/AppData/Local/Microsoft/WindowsApps/python3.12.exe c:/Users/aditn/OneDrive/Desktop/CS634/
ompare_algorithms.py

CS634 Midterm Project
Name: Adit Nuwal
UCID: an238
Email: an238@njit.edu

Enter the minimum support value (between 0 and 1): 0.1
Enter the minimum confidence value (between 0 and 1): 0.1

Processing dataset 1: dataset_1.csv

Apriori algorithm took 0.03210580 seconds.
Brute-force algorithm took 0.10630510 seconds.
We found 534 frequent itemsets.
Generated 1501 association rules.

Here are the top 5 strongest association rules:
{'Fish', 'Bananas'} => {'Rice'} | Confidence: 1.00, Support: 0.25
{'Chicken'} => {'Bananas'} | Confidence: 1.00, Support: 0.20
{'Fish', 'Lettuce'} => {'Rice'} | Confidence: 1.00, Support: 0.20
{'Coffee', 'Rice'} => {'Bananas'} | Confidence: 1.00, Support: 0.20
{'Lettuce', 'Bananas'} => {'Rice'} | Confidence: 1.00, Support: 0.20

Processing dataset 2: dataset_2.csv

Apriori algorithm took 0.01289370 seconds.
Brute-force algorithm took 0.09541580 seconds.
We found 319 frequent itemsets.
Generated 761 association rules.

Here are the top 5 strongest association rules:
{'Beef', 'Eggs'} => {'Potatoes'} | Confidence: 1.00, Support: 0.20
{'Cheese'} => {'Cucumber'} | Confidence: 1.00, Support: 0.15
{'Butter'} => {'Potatoes'} | Confidence: 1.00, Support: 0.15
{'Tomatoes'} => {'Yogurt'} | Confidence: 1.00, Support: 0.15
```



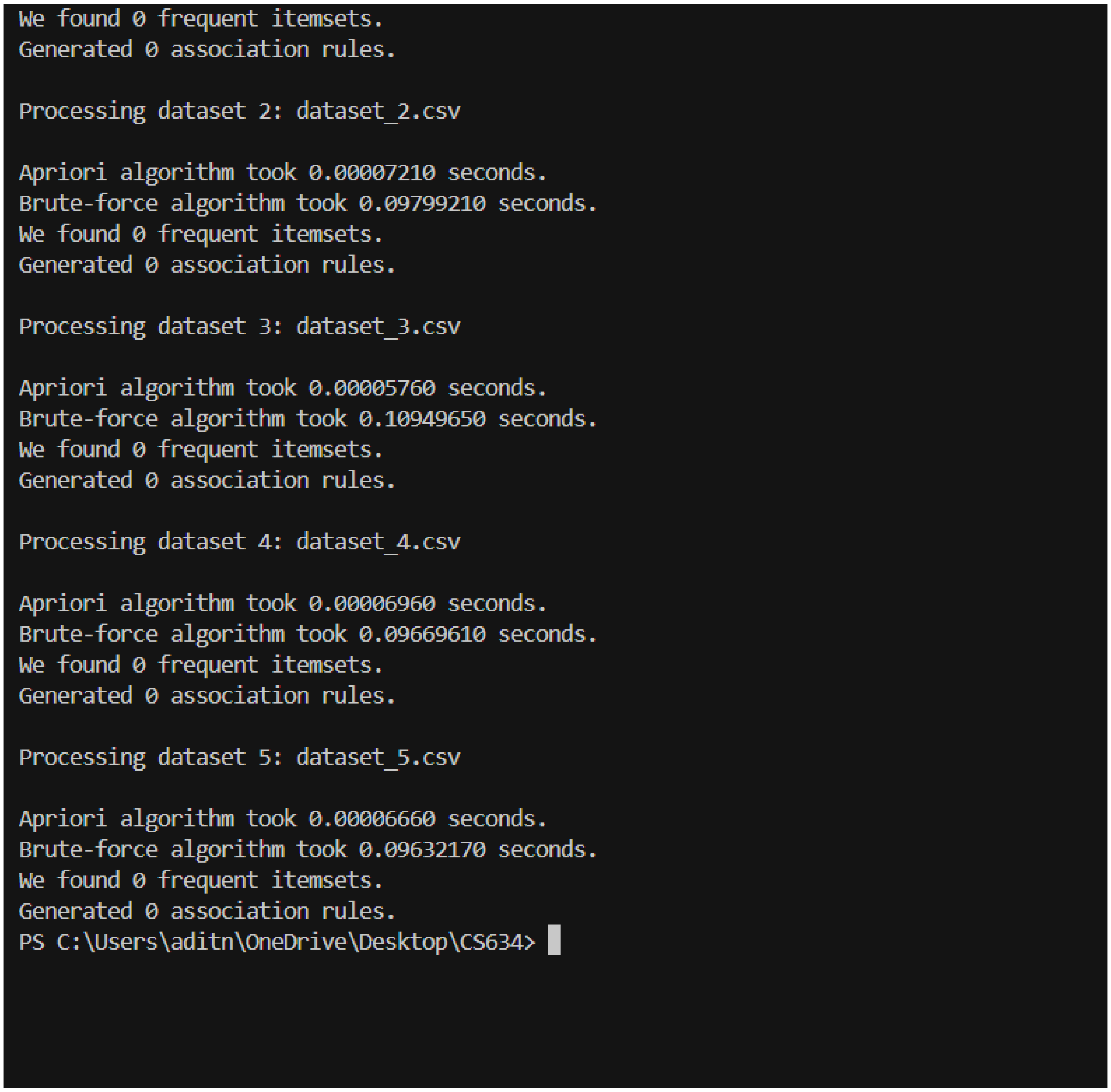
```
{'Potatoes', 'Grapes'} => {'Yogurt'} | Confidence: 1.00, Support: 0.15

Processing dataset 3: dataset_3.csv

Apriori algorithm took 0.02463680 seconds.
Brute-force algorithm took 0.10866350 seconds.
We found 570 frequent itemsets.
Generated 1804 association rules.

Here are the top 5 strongest association rules:
{'Pepper'} => {'Onions'} | Confidence: 1.00, Support: 0.20
{'Yogurt', 'Cucumber'} => {'Chicken'} | Confidence: 1.00, Support: 0.15
{'Chicken', 'Cheese'} => {'Salt'} | Confidence: 1.00, Support: 0.15
{'Salt', 'Cheese'} => {'Chicken'} | Confidence: 1.00, Support: 0.15
{'Cheese', 'Grapes'} => {'Eggs'} | Confidence: 1.00, Support: 0.15

Processing dataset 4: dataset_4.csv

Apriori algorithm took 0.00601310 seconds.
Brute-force algorithm took 0.09572970 seconds.
We found 223 frequent itemsets.
Generated 517 association rules.

Here are the top 5 strongest association rules:
{'Tomatoes'} => {'Pork'} | Confidence: 1.00, Support: 0.15
{'Bread', 'Bananas'} => {'Yogurt'} | Confidence: 1.00, Support: 0.15
{'Pepper', 'Carrots'} => {'Pasta'} | Confidence: 1.00, Support: 0.15
{'Pepper', 'Pasta'} => {'Carrots'} | Confidence: 1.00, Support: 0.15
{'Carrots', 'Pasta'} => {'Pepper'} | Confidence: 1.00, Support: 0.15

Processing dataset 5: dataset_5.csv

Apriori algorithm took 0.01163530 seconds.
Brute-force algorithm took 0.09710770 seconds.
We found 312 frequent itemsets.
Generated 718 association rules.
```



```
Generated 517 association rules.

Here are the top 5 strongest association rules:
{'Tomatoes'} => {'Pork'} | Confidence: 1.00, Support: 0.15
{'Bread', 'Bananas'} => {'Yogurt'} | Confidence: 1.00, Support: 0.15
{'Pepper', 'Carrots'} => {'Pasta'} | Confidence: 1.00, Support: 0.15
{'Pepper', 'Pasta'} => {'Carrots'} | Confidence: 1.00, Support: 0.15
{'Carrots', 'Pasta'} => {'Pepper'} | Confidence: 1.00, Support: 0.15

Processing dataset 5: dataset_5.csv

Apriori algorithm took 0.01163530 seconds.
Brute-force algorithm took 0.09710770 seconds.
We found 312 frequent itemsets.
Generated 718 association rules.

Here are the top 5 strongest association rules:
{'Pepper'} => {'Potatoes'} | Confidence: 1.00, Support: 0.15
{'Beef'} => {'Grapes'} | Confidence: 1.00, Support: 0.15
{'Lettuce', 'Cucumber'} => {'Pasta'} | Confidence: 1.00, Support: 0.15
{'Lettuce', 'Pasta'} => {'Cucumber'} | Confidence: 1.00, Support: 0.15
{'Cucumber', 'Bananas'} => {'Pasta'} | Confidence: 1.00, Support: 0.15
```



```
PS C:\Users\aditn\OneDrive\Desktop\CS634> & C:/Users/aditn/AppData/Local/Microsoft/WindowsApps/python3.12.exe c:/Users/aditn/OneDrive/Desktop
ompare_algorithms.py

CS634 Midterm Project
Name: Adit Nuwal
UCID: an238
Email: an238@njit.edu

Enter the minimum support value (between 0 and 1): 0.8
Enter the minimum confidence value (between 0 and 1): 0.8

Processing dataset 1: dataset_1.csv

Apriori algorithm took 0.00006610 seconds.
Brute-force algorithm took 0.09715930 seconds.
We found 0 frequent itemsets.
Generated 0 association rules.

Processing dataset 2: dataset_2.csv

Apriori algorithm took 0.00007210 seconds.
Brute-force algorithm took 0.09799210 seconds.
We found 0 frequent itemsets.
Generated 0 association rules.

Processing dataset 3: dataset_3.csv

Apriori algorithm took 0.00005760 seconds.
Brute-force algorithm took 0.10949650 seconds.
We found 0 frequent itemsets.
Generated 0 association rules.

Processing dataset 4: dataset_4.csv

Apriori algorithm took 0.00006960 seconds.
Brute-force algorithm took 0.09669610 seconds.
We found 0 frequent itemsets.
Generated 0 association rules.
```



```
We found 0 frequent itemsets.
Generated 0 association rules.

Processing dataset 2: dataset_2.csv

Apriori algorithm took 0.00007210 seconds.
Brute-force algorithm took 0.09799210 seconds.
We found 0 frequent itemsets.
Generated 0 association rules.

Processing dataset 3: dataset_3.csv

Apriori algorithm took 0.00005760 seconds.
Brute-force algorithm took 0.10949650 seconds.
We found 0 frequent itemsets.
Generated 0 association rules.

Processing dataset 4: dataset_4.csv

Apriori algorithm took 0.00006960 seconds.
Brute-force algorithm took 0.09669610 seconds.
We found 0 frequent itemsets.
Generated 0 association rules.

Processing dataset 5: dataset_5.csv

Apriori algorithm took 0.00006660 seconds.
Brute-force algorithm took 0.09632170 seconds.
We found 0 frequent itemsets.
Generated 0 association rules.
PS C:\Users\aditn\OneDrive\Desktop\CS634>
```

Conclusion

- Efficiency of Apriori Algorithm: The results clearly show that the Apriori algorithm significantly reduces computational complexity by leveraging the anti-monotonicity property, which prunes non-frequent itemsets early in the process.
- Limitations of Brute Force Approach: The brute force method, while functionally correct, exhibits an exponential increase in computation time as the number of items and transactions grow. This method becomes impractical for large-scale datasets.

- Performance Comparison: The execution time analysis validates the superiority of the Apriori algorithm, which requires considerably less computation than the brute force method.
- Practical Applications: This project simulates real-world retail transaction analysis, demonstrating how businesses can extract meaningful insights from customer purchase data using association rule mining.

This project successfully demonstrates how association rule mining can be used to discover patterns in transaction data. The Apriori algorithm is computationally efficient compared to the Brute Force approach.